MethodSCRIPT Examples - Android

Last revision: August 21, 2019

www.palmsens.com

1.1 Contents:

The examples in the */MethodSCRIPTExamples_Android* folder demonstrate basic communication with the EmStat Pico from an Android device. The examples show how to connect to the device (using USB and Bluetooth), send a MethodSCRIPT to the device, run measurements on the device, read and parse the measurement data packages from the device using Java.

1.2 Examples:

1.2.1 Example 1: Basic MethodSCRIPT Example (*methodSCRIPTExample*)

This example demonstrates how to implement USB serial communication using the Java d2xx library to

- Establish a connection with the EmStat Pico
- Send a MethodSCRIPT to the EmStat Pico
- Read and parse measurement data packages from the EmStat Pico
- Abort the MethodSCRIPT

This does not include error handling, method validation etc.

1.2.2 Example 2: Bluetooth Example (*msBluetooth*)

This example demonstrates how to communicate with the EmStat Pico from an Android device using a Bluetooth connection.

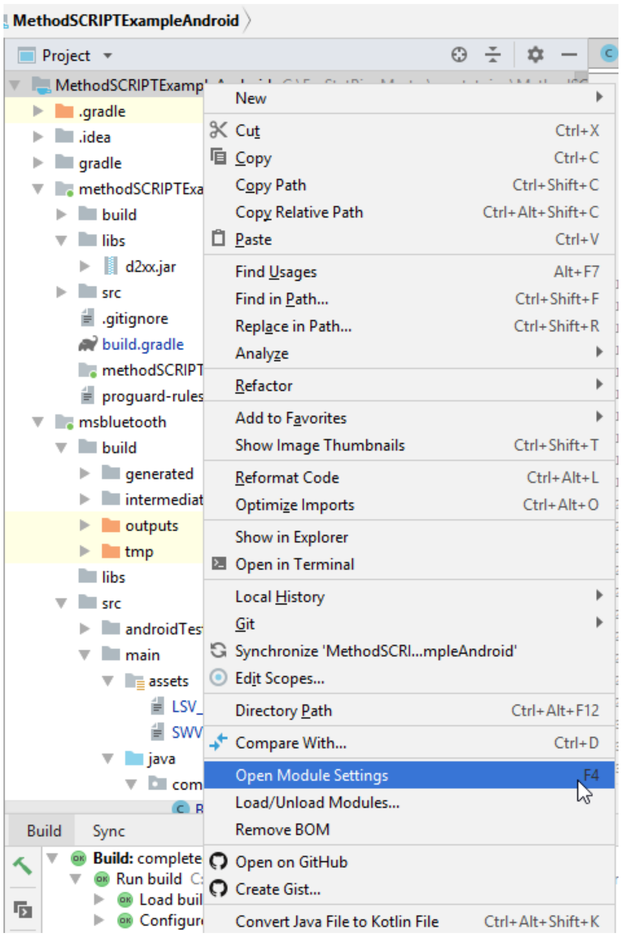
Hardware Setup
In order to connect via Bluetooth, make sure that the UART switch block SW4 on the EmStat Pico development board has the switches for Bluetooth 5 and 6 turned on. Also, the UART Switch block SW3 should have the switches for PICO Bluetooth 5 and 6 turned on.

1.3 Configuring the IDE (Android Studio)

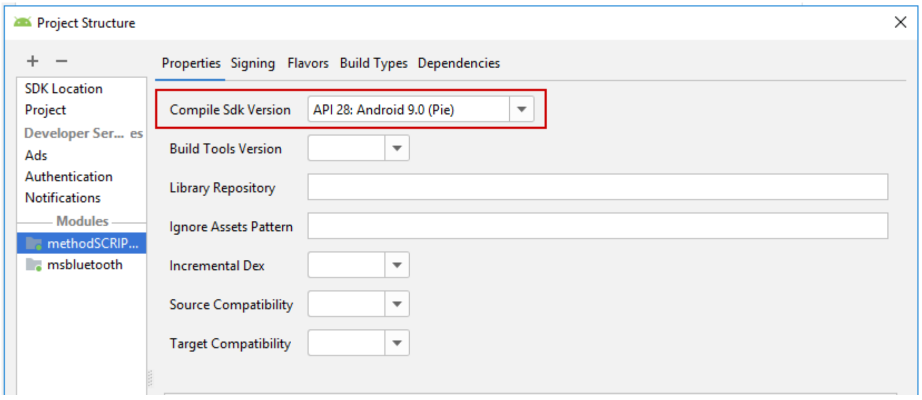
The example projects in the SDK are built using Android Studio 3.3.1. To get started with developing apps on Android Studio, please follow the installation instructions at:
https://developer.android.com/studio/install

1.3.1 Android Studio:

When trying to deploy the example project from Android Studio, if the default settings do not work, please make sure that the following fields are configured under *project -> Open Module Settings*.



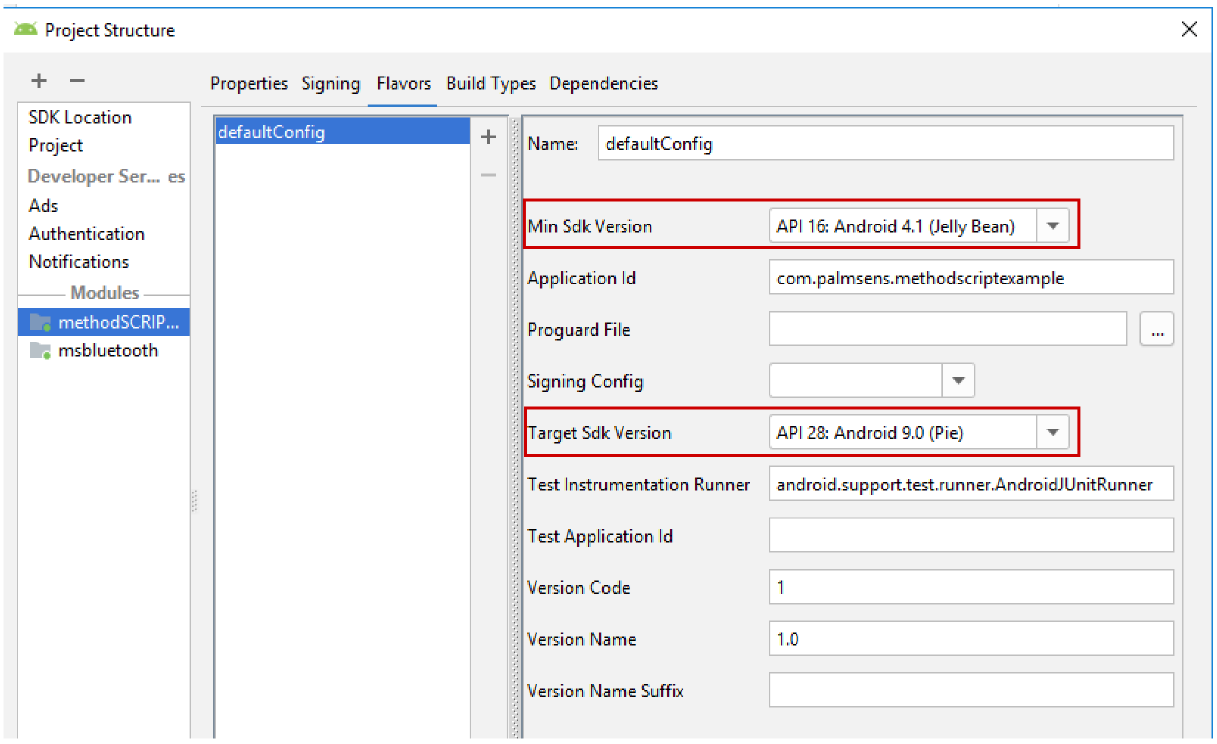
Compile using Android version: Use Latest Platform (API level 28 (Android 9.0 (Pie)) at the time of writing, yours may be higher/newer)



Compile Sdk Version: API 28: Android 9.0 (Pie)
Build Tools Version:
Library Repository:
Ignore Assets Pattern:
Incremental Dex:
Source Compatibility:
Target Compatibility:

Minimum Android to target: API level 16 (Android 4.1 (Jelly Bean))

Target Android version: Use the same version as the version used in Compile Sdk Version.



Name: defaultConfig
Min Sdk Version: API 16: Android 4.1 (Jelly Bean)
Application Id: com.palmsens.methodscriptexample
Proguard File:
Signing Config:
Target Sdk Version: API 28: Android 9.0 (Pie)
Test Instrumentation Runner: android.support.test.runner.AndroidJUnitRunner
Test Application Id:
Version Code: 1
Version Name: 1.0
Version Name Suffix:

Please make sure (using Tools -> SDK manager), that the newest version of the Android SDK is installed else update it to the newest.

Note: Do not forget to enable “USB Debugging” on the Android device while deploying the app.

1.4 Communications

1.4.1 Connecting to the device

The Android device can either use either Bluetooth or USB serial communication with the EmStat Pico.

1.4.1.1 USB Serial communication

The *MethodSCRIPTExample* module demonstrates the USB serial communication using the Java d2xx library (downloaded from here). The D2xx.jar is placed in the libs folder in the *MethodSCRIPTExample* module. The D2xxManager and FT_Device classes are used in the example to communicate with the EmStat Pico.

```
import com.ftdi.j2xx.D2xxManager;
import com.ftdi.j2xx.FT_Device;

ftD2xxManager = D2xxManager.getInstance(this);
```

The Android device needs USB permissions to communicate with USB devices. These permissions have to be included in the AndroidManifest.xml.

```
<uses-feature android:name="android.hardware.usb.host" />
<uses-permission android:name="android.permission.USB_PERMISSION" />
```

In order to discover a particular USB device, an intent filter has to be specified in the AndroidManifest.xml to filter for the android.hardware.usb.action.USB_DEVICE_ATTACHED intent. Along with this intent filter, a resource file that specifies properties of the USB device, such as product and vendor ID has to be included as meta-data.

```
<activity
android:name="com.palmsens.methodSCRIPTExample.MethodSCRIPTExample"
android:launchMode="singleTask">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <action
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
        <action
android:name="android.hardware.usb.action.USB_DEVICE_DETACHED" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <meta-data
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
android:resource="@xml/device_filter"/>
</activity>
```

The device_filter resource file contains the properties of the USB device (Product ID and Vendor ID) as shown below.

```
<resources>
    <!-- 0x0403 / 0x6015: FTDI FT231X -->
    <usb-device vendor-id="1027" product-id="24597" />
</resources>
```

Upon discovering the device when USB is attached, the device can be opened using the ftD2xxManager instance created at the beginning as shown below.

```
ftDevice = ftD2xxManager.openByIndex(this, 0);
```

The baud rate of the EmStat Pico has to be set to 230400.

Upon connecting to the device, the version command is sent to verify if the device is EmStat Pico. A new thread is started which keeps receiving the measurement packages until the device is disconnected.

1.4.1.2 Bluetooth communication

The msBluetooth example demonstrates the communication between Android device and EmStat Pico over Bluetooth. The Android device needs the following Bluetooth permissions to be specified in the AndroidManifest.xml

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

On Android devices with API level 23 (Android 6.0 (Marshmallow)) and above, the location permissions needed for Bluetooth have to be requested at run time as shown in the code snippet below.

```
if (checkSelfPermission(Manifest.permission_group.LOCATION) ==
PackageManager.PERMISSION_DENIED) {
    this.requestPermissions(new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION}, REQUEST_CODE_LOCATION);
}
```

In order to establish a Bluetooth connection, a new BluetoothConnectionService instance is created in the main activity, which in turn starts a separate thread to connect and communicate with the EmStat Pico.

A BluetoothSocket instance for the selected device can be created using the createInsecureRfcommSocketToServiceRecord of the BluetoothDevice class as shown below. The UUID of the device can be found as shown in the code below.

```
ParcelUuid[] uuid = mSelectedDevice.getUuids();
mSelectedUUID = uuid != null ? uuid[0].getUuid() : DEFAULT_UUID;
```

The default SPP UUID for a Bluetooth serial board is used in this example as shown below.

```
private static final UUID DEFAULT_UUID = UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB");
```

Upon successful connection of the Bluetooth socket, the read thread continues to read from the instream when data is available.

1.4.2 Sending a MethodSCRIPT

The MethodSCRIPT can be read from a text file stored in the Android device and sent to the EmStat Pico. In this example the MethodSCRIPT files are stored in the *assets* directory and read as shown below.

```
bufferedReader = new BufferedReader(new
InputStreamReader(getAssets().open(SCRIPT_FILE_NAME)));
```

The MethodSCRIPT can be sent to the EmStat Pico through USB as shown in the code below.

```
bytesWritten = ftDevice.write(writeByte, script.length());
```

The MethodSCRIPT can be sent to the EmStat Pico over Bluetooth using the Bluetooth socket out stream as shown in the code below.

```
mOutStream.write(bytes);
```

1.4.3 Receiving the measurement packages

Once the MethodSCRIPT is sent to the device, the measurement packages can be read continuously from the device.

The available data, in the case of USB connection, is read one character at a time as shown in the code below.

```
readSize = ftDevice.getQueueStatus();   //Retrieves the size of the
available data on the device
if (readSize > 0) {
    ftDevice.read(rbuf, 1);
    ....
}
```

The available data, in the case of Bluetooth connection, is read one character at a time as shown in the code below.

```
if (mInStream != null && mInStream.available() > 0) {
    readSize = mInStream.read(rbuf, offset, 1);   //Reads a character from
the socket instream
        ....
}
```

The measurement packages read over Bluetooth are posted back on the handler to be sent back to the main activity for further parsing.

1.4.4 Parsing the measurement packages

The measurement data packages can be parsed further to obtain the actual data values. Here's a set of data packages received from a Linear Sweep Voltammetry (LSV) measurement on a dummy cell with 10kOhm resistance.

```
e\n
M0000\n
Pda7F85F3Fu;ba48D503Dp,10,288\n
Pda7F9234Bu;ba4E2C324p,10,288\n
Pda806EC24u;baAE16C6Dp,10,288\n
Pda807B031u;baB360495p,10,288\n
*\n
\n
```

While parsing a measurement package, various identifiers are used to identify the type of package. For example, In the above sample,

1. 'e' is the confirmation of the "execute MethodSCRIPT" command.
2. 'M' marks the beginning of a measurement loop.
3. 'P' marks the beginning of a measurement data package.
4. "*\n" marks the end of a measurement loop.
5. "\n" marks the end of the MethodSCRIPT.

The data values to be received from a measurement can be sent through 'pck*'* commands in the MethodSCRIPT. Most techniques return the data values Potential (set cell potential in V) and Current (measured current in A). These can be sent with the MethodSCRIPT.

In case of Electrochemical Impedance Spectroscopy (EIS) measurements, the following *variable types* can be sent with the MethodSCRIPT and received as measurement data values.

- Frequency (set frequency in Hz)
- Real part of complex Impedance (measured impedance Ohm)
- Imaginary part of complex Impedance (measured impedance in Ohm)

The following metadata values can also be obtained from the data packages, if present.

- CurrentStatus (OK, underload, overload, overload warning)
- CurrentRange (the current range in use)
- Noise (Noise)

1.4.4.1 Parsing the measurement data packages

Each measurement data package begins with the header 'P' and is terminated by a '\n'. The measurement data package can be split into data value packages based on the delimiter ';'. Each of these data value packages can then be parsed separately to get the actual data values.

The type of data in a data package is identified by its variable type:

- The potential readings are identified by the string *da*
- The current readings are identified by the string *ba*
- The frequency readings are identified by the string *dc*
- The real impedance readings are identified by the string *cc*
- The imaginary impedance readings are identified by the string *cd*

For example, in the sample package seen above, the *variable types* are
da7F85F3Fu - *da* Potential reading and
ba48D503Dp,10,288 – *ba* current reading.

The following 8 characters hold the data value. The data value for the current reading (8 characters) from the above sample package is *48D503Dp*.

The SI unit prefix from the package can be obtained from the parameter value at position 8

In the above sample package, the unit prefix for current data is ‘p’ which is 1e-12 A.

After obtaining variable type and the data values from the package, the metadata values can be parsed, if present.

1.4.4.2 Parsing the metadata values

The metadata values are separated based on the delimiter ‘,’ and each of the values is further parsed to get the actual value.

The first character of each metadata value metaData[0] identifies the type of metadata.

‘1’ – status
‘2’ – Current range index
‘4’ - Noise

The metadata status is a 1 character hexadecimal bit mask.

For example, in the above sample, the available metadata values for current data are,
10,288. The first metadata value is 10.

1 – metadata status – 0 indicates OK.

The metadata type current range is 2 characters long hexadecimal value. If the first bit is high (0x80), it indicates a high-speed mode current range. The hexadecimal value can be converted to int to get the current range.

For example, in the above sample, the second metadata available is 288.

2 – indicates the type – current range
88 – indicates the hexadecimal value for current range index – 1mA. The first bit 8 implies that it is high-speed mode current range.

Sample output:

Here’s a sample measurement data package from a LSV measurement on a dummy cell with 10 kOhm resistance and its corresponding output.

```
Pda7F85F3Fu;ba4BA99F0p,10,288

Output:                E (V) = -4.999E-01
            i (A) = -4.999E-01
            Status : OK
            CR: 1mA (High speed)
```